一种改进的云存储系统容错机制*

聂瑞华[a,b], 张科伦[b], 梁　军[a]
(华南师范大学 a. 网络中心; b. 计算机学院, 广州　510631)

摘　要: 在云存储平台下,提出了一种基于访问统计的自适应容错机制 SFMAF,该机制通过近似最近最少使用算法维护一张文件访问频率表来自适应调整容错方式。SFMAF 对于常读取的文件采用副本冗余机制;对于不常读取的文件采用 Reed-Solomon(RS)纠删码容错机制。实验结果表明,SFMAF 相对于副本冗余机制,在 CPU 和内存使用率可接受范围内的增加上,减少了系统内部数据的传输流量,即减少了系统存储空间。

关键词: 云存储; 纠删码; 文件访问频率表; 自适应; 容错机制
中图分类号: TP301.6　　**文献标志码**: A　　**文章编号**: 1001-3695(2013)12-3724-05
doi:10.3969/j.issn.1001-3695.2013.12.053

Improved fault-tolerance mechanism in cloud storage system

NIE Rui-hua[a,b], ZHANG Ke-lun[b], LIANG Jun[a]
(*a. Network Center, b. School of Computer, South China Normal University, Guangzhou* 510631, *China*)

Abstract: This paper presented a self-adaptive fault-tolerance mechanism based access frequency(SFMAF) in cloud storage platform. This mechanism was implemented by maintaining a file access frequency table using approximate least recently used algorithm. If the file was read frequently, it used replica redundancy mechanism. If the file was seldom read, it used RS erasure code mechanism. Experimental results show that, comparing to replica redundancy mechanism, SFMAF increases CPU and memory utilization within the acceptable range, and reduces the data transmission flow within the system, that reduces the storage space in system.

Key words: cloud storage; erasure code; file access frequency table; self-adaptive; fault-tolerant mechanism

0　引言

随着数据量的急剧增长,数据存储需求对存储系统的可靠性、吞吐性能提出了更高要求。分布式存储系统能够分散数据服务器的压力,获得较高的吞吐量,提供高可靠性、高性能的存储系统,它已经成为当前服务供应商常用的存储技术,其中云存储就是一个很典型的应用。

大部分云存储平台[1,2]的容错机制为副本冗余机制,该系统实现简单,能达到服务器负载均衡的目的,有较高的吞吐量,但副本备份需要消耗大量的存储空间。有学者提出采用纠删码容错机制作为云存储系统的容错机制,采用这种方法能够减少备份所消耗的存储量,并且能够提供较高的可靠性,但会降低服务器负载均衡能力。

在实际的生产活动中,云存储系统中并不是所有的文件都处于经常被读取的状态,有相当一部分文件很少被读取或者仅作为备份供以后查看。针对这种情况,本文结合了上述两种方法,在文件系统中加入一个由基于近似最近最少使用算法维护的文件频率表。根据文件读取频率,自适应地对相应文件的备份策略在副本冗余机制和纠删码容错机制之间进行转换,在系统资源可接受范围内的增加上,减少了系统存储空间。

1　Hadoop 分布式文件系统简述

Hadoop 项目是 Apache 基金组织的一个项目,其目的是实现一个运行在大数量廉价的普通部件上的数据密集型的、可靠的、可扩展的分布式计算框架。Hadoop 是一个开源项目,已经成为一个非常流行的分布式数据密集型应用的快速解决方案模型。很多公司都对 Hadoop 项目有极大的兴趣,包括 Yahoo!、Facebook、twitter、AOL、eyealike 等。其中,Yahoo! 已经将 Hadoop 部署在由 4 000 个节点搭建的集群上,用来实现商业应用[3]。

1.1　Hadoop 分布式文件系统的架构

Hadoop 分布式文件系统(hadoop distributed file system, HDFS),它是 Hadoop 框架的存储基础层,是 Hadoop 架构中的一个关键组成部分,实现了框架对于输入、输出数据的存储、管理和差错控制功能。HDFS 具有很高的可靠性、可扩展性、文件吞吐率及移植性,适用于廉价的普通部件的大规模集群上的分布式系统。

HDFS 是典型的主、从体系结构,它主要由名称节点(nameNode)、辅助名称节点(secondary NameNode)、数据节点(dataNode)三部分组成[4~6]。系统中一般只有一个名称节点,它记录文件系统的元数据、组织集群中数据节点的工作并且协调客

收稿日期: 2013-03-16; **修回日期**: 2013-04-28　　**基金项目**: 国家教育部互联网应用创新开放平台示范基地项目(教技发中心函[2012]120 号);国家自然科学基金资助项目(61370178);国家教育部—中国移动科研基金资助项目(MCM20121051)

作者简介:聂瑞华(1963-),男,江西樟树人,教授,硕士,主要研究方向为计算机网络及应用、云计算与云存储、信息系统与信息集成等(nrh@scnu.edu.cn);张科伦(1986-),男,硕士研究生,主要研究方向为计算机网络及应用;梁军(1983-),男,工程师,硕士,主要研究方向为计算机网络及应用、Linux 应用系统开发.

户端读取文件的工作。辅助名称节点对名称节点中的元数据进行备份,当名称节点失效时辅助名称节点将很快替代名称节点的工作,保证系统的正常运行。数据节点用于存储文件块,进行数据管理。HDFS 集群中包含多个数据节点,并可以根据需要扩展集群规模,任何可以连接到互联网、并且运行有 Linux 操作系统的主机都能注册成为数据节点。

1.2　Hadoop 分布式文件系统的容错机制

Hadoop 在设计之初就将框架定位为在廉价的普通硬件上运行的系统。HDFS 把硬件出错当做是一种常态,它采取了相应的容错机制来保证系统的健壮性。

HDFS 的容错策略采取了副本冗余策略:a) HDFS 对于每一个数据块都进行了副本的复制,名称节点根据指定的副本数,控制数据的复制,即根据副本数的要求,数据节点间以管道的方式传输数据库形成冗余副本;b) 对于副本数的维护,每个数据节点都会定期向名称节点发送心跳(heartbeat)信息,如果当名称节点检测到某个数据节点没有成功发送心跳信息,那么名称节点将会标记这个数据节点为失效,并且不会向该节点发送 I/O 请求。在上述情况下,与失效的数据节点相关联的数据块已经丢失,名称节点会定期检测数据块的副本数,如果在检测过程中名称节点发现某些数据块的副本数少于指定的副本数,那么它将选择适当的时机向尚保有该数据块的数据节点发送复制文件块的命令,从而保证副本数与设置参数的一致性。

2　Reed-Solomon (RS)纠删码

近年来,许多研究者都将纠删码作为分布式系统中的冗余机制来提高服务的可靠性。纠删码的基本思想是[7]对 k 块源数据进行编码,产生 n 块编译的目标编码($n>k$)。在这 n 块编码中任意获取 k 块就能够将 k 块源数据还原出来。在 n 块编码的编码组中,最多丢失 $n-k$ 个块也不会影响数据的还原,称这种编码为(n, k)纠删码。

RS 纠删码是 Reed 等人[8,9]提出的。RS 类纠删码根据生成矩阵的不同分为两类,即范德蒙 RS 码和柯西 RS 码。RS 编码是唯一可以满足任意的数据磁盘数目 n 和冗余磁盘数目 m 的 MDS(maximum distance separable)的编码方法。RS 纠删码算法中主要考虑几个方面[10]:使用范德蒙德矩阵计算,从而生成、维护校验数据;利用高斯消元从错中从而生成、维护校验数据;利用高斯消元从错误中恢复数据;使用伽罗瓦域进行有限域上的运算。

1)计算校验和　定义 $X=(x_1,x_2,\cdots,x_m)$ 是元数据包向量,$Y=(y_1,y_2\cdots,y_n)$ 是校验和数据包向量,$G_{m\times n}$ 为线性纠删码的生成矩阵,$G_{i,j}=j^{i-1}$ 为范德蒙德矩阵,即

$$G=\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & n \\ \vdots & \vdots & \vdots & & \vdots \\ 1^{m-1} & 2^{m-1} & 3^{m-1} & \cdots & n^{m-1} \end{pmatrix} \tag{1}$$

那么,$Y=XG$ 为

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & n \\ \vdots & \vdots & \vdots & & \vdots \\ 1^{m-1} & 2^{m-1} & 3^{m-1} & \cdots & n^{m-1} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}=\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \tag{2}$$

2)维护检验和　当数据 X 改变成 X' 时,随之其对应的校验和数据也应改变。具体步骤为修改后数据减去原数据后重新计算校验和,将新计算出的校验和加上原校验和得到最终校验和,公式为

$$Y'=Y+(X'-X)G \tag{3}$$

3)利用高斯消元法恢复失效数据　定义矩阵 $A=\begin{pmatrix} I \\ G \end{pmatrix}$,向量 $E=\begin{pmatrix} X \\ Y \end{pmatrix}$。矩阵 I 为单位矩阵。有 $AX=E$,即

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \cdots & n^{m-1} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}=\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{4}$$

当向量 E 中的数据丢失时,应将矩阵 A 中对应的行删除得到新的向量 E' 以及矩阵 A',则有 $A'X=E'$。假设向量 E 中有 m 项的数据丢失,则将矩阵 A 中对应的 m 行删除。矩阵 G 是一个范德蒙德矩阵,它的任意行之间是线性无关的,所以从矩阵 A 中任取 p 行所得到的矩阵的任意行之间都是线性无关的。则矩阵 A' 是非奇异的,即矩阵 A' 是可逆的,向量 X 的值可以通过 $A'X=E'$ 利用高斯消元法计算得出。因此,当有小于等于 m 个数据块丢失时,能通过该方法有效地恢复数据。

4)利用伽罗瓦域进行有限域上的运算,保证计算结果不发生溢出[11]　伽罗瓦域对算术运算是闭包的,它的加减法运算都是异或运算。在进行乘除法运算时,可以将对于生成元素的指数直接相加减取模。

3　基于访问统计的自适应容错机制 SFMAF 及其实施

3.1　基于访问统计的自适应容错机制

3.1.1　基于访问统计的自适应容错机制的设计思路

基于访问统计的自适应容错机制(self-adaptive fault-tolerant mechanism based access frequency,SFMAF)的设计目的是自适应地调整文件的容错机制,从而在达到节省存储空间的同时,兼顾文件访问的负载均衡与读取速度。在 SFMAF 的设计中,利用近似最近最少访问算法在系统中维护了一张文件访问频率表,在频率表中的文件为最近被读取或者最近被读取频率较高的文件的集合,其容错机制采取副本冗余容错机制。通过对文件进入表的时长和进入表后被访问次数的比值进行计算来将最近最少访问的文件移出频率表,同时将该文件的容错机制由副本冗余机制转换为纠删码冗余机制。

在何时将文件加入访问频率表,何时将其从表中移除是维护该文件访问频率表的关键。

在实际的应用中,数据被存储到云储存系统后通常有以下两种情况:a)对于第一次写入系统的数据,在短期内需要被多次读取,数据写入后进行次数较少的读取或供以后查看;b)对于已经在系统中的文件,其有一段时间被读取的频率较小,最近被频繁读取,文件在前一段时间内经常被读取,但最近读取频率逐渐下降。对于情况 a),在用户没有明确指定的情况下,系统无法有效地判断该文件在近期内是否会被读取,所以系统在这种情况下默认文件写入后最近将要被多次读取,故应采取副本冗余机制来应对可能发生的多客户端、多次文件读取。对

于情况 b),当文件前一段时间被读取的频率很小、最近被频繁读取时,在这种访问频率上升的趋势下,系统认为文件将在近期被多次读取,故同样采取副本冗余机制存储数据块;对于文件前一段时间内经常被读取(此时文件的容错机制仍为副本冗余机制)、最近的访问频率逐渐下降的情况下,根据系统与先设定的算法在适当的情况下将文件的容错机制转换为副本冗余机制,以减小保存数据块的存储空间。

评价一个文件的最近访问频率采用了两个参数,即在文件频率表中的时间段内被访问的次数及进入频率表的时长。采用两个参数作为评价标准是为了避免采用单一参数作为评价标准时容易发生的比较极端的情况:a)单独采用读取次数作为评价标准时,有一个文件在较早前的某一个时间段被非常多地读取,其读取次数超过频率表中大部分的文件,但是该文件在较长的一段时间几乎没有被读取,而其他的大部分文件在短时间内的被读取次数很难达到这个文件的读取次数,所以该文件不会被置换出频率表,占用了频率表的空间;b)单独采用文件进入频率表的时间作为评价标准时,有一个文件被加入频率表的时间较长,在加入频率表后的时间段内持续被多次读取,当有较多新读取的文件加入频率表时,进入访问频率表时间较长又持续被读取的文件将被置换出频率表。按照其读取趋势,被置换出的文件近期还有可能被读取。综上所述,采取两个参数来作为评价标准能够平衡上面两种情况,实现一个比较合理的策略。

3.1.2 基于访问统计的自适应容错机制的策略

根据 3.1.1 节对存储系统中文件状态的分析,在该 SFMAF 机制中,使用了如下策略:

a)当一个文件首次写入时,系统采取 HDFS 的默认方式存储数据,即副本冗余方式。采取这种存储方式的考虑是:系统不能判断文件在写入系统后是否要被多次读取,但是文件初次写入后被读取的可能性较大。故在文件会被多次读取的情况下,采用副本冗余机制能够有效地实现负载均衡以及提高读取效率的目的。在文件写入后未被经常读取的情况下,根据系统中的自适应调整机制文件的容错方式会很快被调整为纠删码容错机制。

b)在系统中维护一张文件访问频率表。当有文件进行读取、文件名在频率表上时,对其访问次数加 1;当文件访问次数达到指定数时,认为该文件最近经常被访问,更改该文件的容错方式为副本冗余容错机制。

c)当有文件进行读取、文件名不在频率表上、文件频率表已满时,首先对表中各文件的时长/次数比进行比较,找出时长/次数比最高(访问频率最低)的文件记录,将其置换出频率表;然后将被读取的文件的信息加入访问频率表,访问次数置为 1, 记录其进入文件表的时间戳。

d)当有文件进行读取、文件名不在频率表上、文件频率表未满时,将被读取的文件的信息加入访问频率表,访问次数置为 1, 记录其进入文件表的时间戳。

e)置换出频率表的文件,为了避免频繁地进行纠删码机制转换的计算,待下次扫描频率表时判断该移除的文件是否又进入频率表。如果未在频率表中出现,容错机制转换为纠删码容错机制;如果进入频率表,则不转换。

f)定期清理文件频率表,以时间戳作为排序依据,将访问次数大于 1 并且小于文件访问指定次数的文件清理出文件访问频率表,以控制其大小。首先遍历频率表,在容错机制仍为纠删码容错机制的文件信息中找出进入频率表时长超过时长阈值的文件,接着将符合条件的文件信息从频率表中移除。时长阈值可设定为清理频率表时间间隔的 2 ~ 3 倍,即表示文件在频率表经历了 2 ~ 3 次的清理后,该文件的容错机制仍未发生转变、访问频率并不高,可不对其访问次数进行统计并移除。

3.1.3 算法描述

假设频率表内的文件对应读取次数为 t,容错机制状态位为 s,进入表的时间戳为 ts,文件访问指定次数为 num。在进行表内文件替换时,记当前时间戳为 tc,则相应文件在表内持续停留时长为 $d = tc - ts$。

读取文件时,自适应调整容错的算法描述如下:

```
Dim t As Integer ‘定义文件对应读取次数’
Dim ts As String ‘定义进入表的时间戳’
Dim tc As String ‘定义在进行表内文件替换时记当前时间戳’
Dim d As float ‘定义相应文件在表内持续停留时长’
Dim num As Integer ‘定义文件访问次数达到指定数’
Dim s As Boolean ‘定义容错机制状态位’
读取文件;
if(文件名在频率表中)THEN
  t = t + 1;
  if(t >= num)THEN
    将对应文件的备份方式更改为副本冗余机制,且设状态位 s = 1;
  end if
  返回文件相关信息;
else
  if(文件频率表已满)THEN
    将 max(d/t)对应文件置换出表(在等待一定的时长后,转换出的文件未再次出现在频率表中,将其转换为纠删码容错机制);
  end if
  将新读取文件的相关信息置入文件频率表,且设状态位 s = 0;
  记录新读取文件进入频率表的时间戳;
  新读取文件设 t = 1;
end if
```

3.2 SFMAF 的纠删码容错的实施

3.2.1 HDFS 中部分类的介绍

在 HDFS 中,表示文件信息的类为 INode 类。INode 类是一个抽象类,HDFS 中主要有两个类实现继承了 INode 类:INodeDirectory 类以及 INodeFile 类,分别表示路径和文件。名称节点在内存中维护由 INode 子类组成的树型结构元数据来描述整个文件系统的结构。此外,INodeFile 在 BlockInfo 数组中保存了文件的块信息,故通过 INodeFile 能够获取“文件—块”的映射关系。另外还有两个类:INodeDirectoryWithQuota 类是 INodeDirectory 的子类,该子类描述了一个限定了容量的路径;INodeFileUnderConstruction 类是 INodeFile 的子类,该子类描述正在构造的文件,当创建文件时,目录系统会维护对应的信息。

BlockInfo 类是 HDFS 系统中文件按块存储的关键描述部分,它继承自 Block 类。Block 类是一个抽象类,它是一个数据块或者数据存储单元的抽象描述,其中包含了块 ID、块大小、块的时间戳等基本信息。而在 BlockInfo 类中的主要信息包括该数据块属于哪一个文件、该数据块保存在哪些数据节点中、该数据块的上一个数据块、该数据块的下一个数据块。

3.2.2 SFMAF 中纠删码容错的实施

本文在 INodeFile 类中加入了 BlockInfo 信息数组域 checksum_blks,来实现纠删码容错机制。Checksum_blks 中记录了

校验数据块(编码后的数据块)的文件信息。当需要寻找校验数据块时,可以通过 INodeFile 类中的 checksum_blks 找到对应的 BlockInfo 的实例后与选定的数据节点通信获取数据。

通过 BlockInfo 的反向查找能力能够很方便地由一个源数据块找到所有的源数据块和校验数据块的集合。修改后的 INode 类图如图 1 所示。

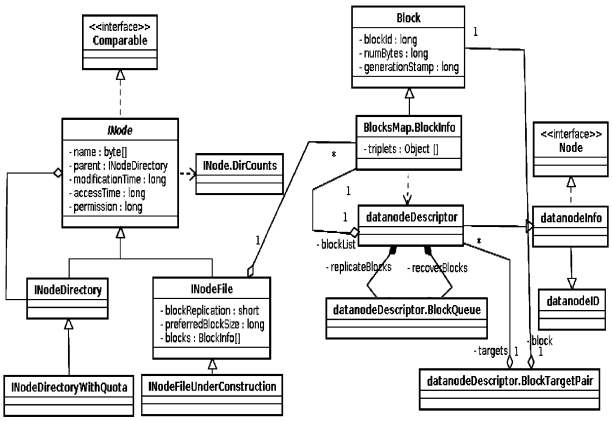

图 1　INode 类图

首先设文件被分为 k 块源数据块,且在 SFMAF 中的校验数据块也为 k,当文件成功写入后,数据块失效时系统恢复数据块有三种情况:

a)文件源数据块发现失效,校验数据块未发现失效块。在这种情况下,校验数据块都未丢失,可以通过纠删码的“解码”算法重新计算出源数据块。名称节点确定失效块后,根据“数据块—数据节点列表”映射表获取相应的 k 块校验数据的数据节点映射列表。将该列表封装在“解码”命令中发送至某一个数据节点。该数据节点在接收到命令后对丢失块进行解码运算,恢复失效数据块。

b)文件源数据块未发现失效,校验数据块发现失效块。在这种情况下,k 块数据块都未丢失,可以直接进行纠删码“编码”算法重新计算校验数据块。名称节点确定失效的校验数据块后,根据“数据块—数据节点列表”映射表获取相应的 k 块源数据块的数据节点映射列表。将该列表封装在“编码”命令中发送至某一个数据节点。该数据节点在接收到命令后对丢失块进行编码运算,恢复失效数据块。

c)文件源数据块与校验数据块均发现失效块。在这种情况下,实现数据恢复的前提是未失效源数据块块数与未失效校验数据块块数之和大于等于文件源数据块分块数 k。在满足上述条件的前提下,名称节点将在尚未失效的源数据块、校验数据块中选取 k 块,根据“数据块—数据节点列表”映射表获取相应的 k 块源数据块的数据节点映射列表,然后将列表随“解码”命令发送给某一个数据节点。数据节点接收命令后进行解码运算,恢复出丢失的数据块。

3.3　SFMAF 中文件访问频率表的实施

在 SFMAF 架构中,为实现文件访问频率表定义了一个 AccessFrequenceMap 类。它的主要作用是维护记录文件访问信息〈文件,文件访问信息(FileAccessInfo 类)〉的哈希文件访问频率表。FileAccessInfo 类中记录了文件信息、文件进入频率表时间戳以及文件被访问次数。AccessFrequenceMap 类主要功能有添加文件访问信息、移除文件访问信息、维护文件访问频率表。

当调用 AccessFrequenceMap 添加文件的方法时,首先检测该文件是否已经存在于频率表中。虽然直接将文件添加进哈希表形式的文件访问频率表也不会产生重复的元素,但是为了统计文件的访问次数,故先判断文件是否在哈希表中。如果文件已经在频率表中,那么通过“键—值”对找到文件对应的 FileAccessInfo 类的实例,并且调用方法增加实例中所记录的文件访问次数。如果文件不在频率表中,则将文件添加进入频率表,并且记录文件加入的时间戳,并设置访问次数为 1。

当调用移除方法时,首先遍历频率表,找出容错机制为副本冗余机制的文件中时长次数比最高的对应记录;然后将该记录从频率表中移除;再返回该文件的 INodeFile 信息,以便调用该方法的类进行进一步的操作。

在 HDFS 中,名称节点对文件系统的存储和管理实际由 FSNamesystem 类来实现。在 FSNamesystem 类中记录了“文件—块”映射表(保存在本来文件系统)、数据块—数据节点列表”(根据数据节点的报告,动态地重建,保存在内存中)、“数据节点—块”映射记录每个数据节点中的所有块等重要信息。

SFMAF 框架在 FSNamesystem 类中还维护了文件访问频率表。文件访问频率表在系统启动后随着文件的读取开始形成并维护。在 FSNamesystem 中,用一个线程持续地定期调用 AccessFrequenceMapMonitor 类中的 checkAccessFrequenceMap 方法。通过该方法能够调用 AccessFrequenceMap 类中维护频率表的方法清理频率表,从而达到了定期清理频率表、控制频率表大小的目的。在文件被读取后,调用 addFileToAFMap 方法将文件向频率表中添加。如果文件的访问次数达到 num,文件调整为副本冗余机制;同时通过 INodeFile 内的校验块列表找到校验块所在的数据节点,并向这些数据节点发送删除指定数据块的命令。

以上是对文件访问频率表的实施策略说明。修改后的 FSNamesystem 类图如图 2 所示。

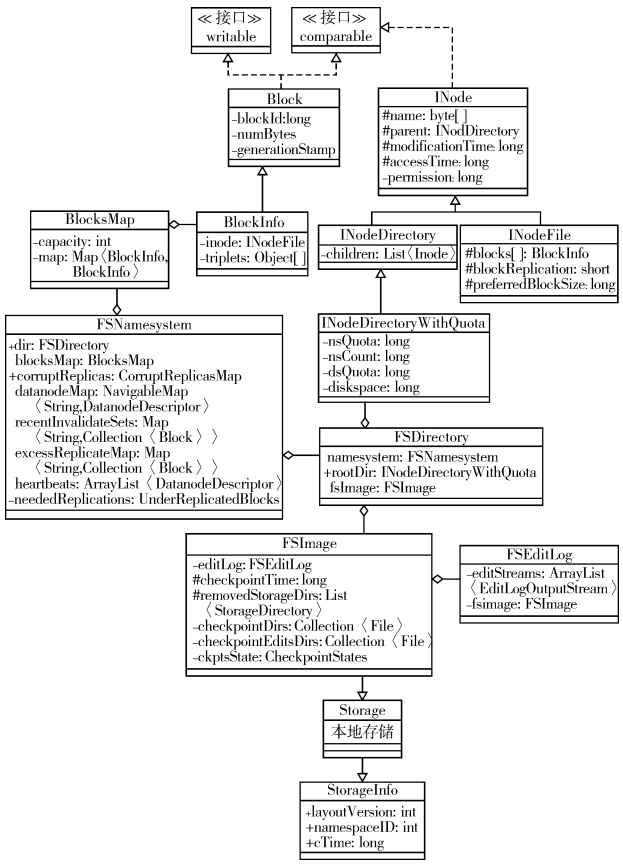

图 2　FSNamesystem 类图

4 实验与分析

网络拓扑结构如图 3 所示。实验由八台服务器、两台交换机、一台笔记本电脑组成,其中:两台服务器作为名称节点和辅助名称节点,六台服务器作为集群的数据节点,笔记本电脑作为客户端对系统进行访问和操作。名称节点服务器的 CPU 主频为 2.4 GHz、内存为 2 GB、硬盘为 60 GB;数据节点服务器的 CPU 主频为 2 GHz、内存为 1 GB、硬盘为 120 GB。

对采用 SFMAF 和完全副本冗余机制的云存储系统进行 24 h 不间断读取、写入文件的模拟比较。在 SFMAF 中,参数 $num=5$, 文件访问频率表的大小设置为 1 000 条记录,文件副本数设为 3。在完全副本冗余机制下,文件副本数设为 3。使用开源软件 Ganglia 对服务器的内存使用情况、CPU 占用率以及网络吞吐量进行监测。图 4 ~ 7 是各项性能指标的比较结果。图 4、5 是内存占用率比较,图 6 是 CPU 占用率比较,图 7 是系统内部数据的传输流量比较。

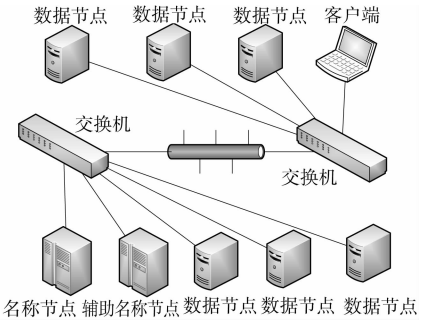

图 3 网络拓扑结构

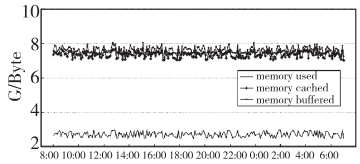

图 4 采用完全副本冗余机制的内存使用情况

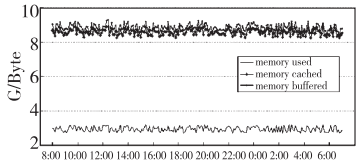

图 5 采用 SFMAF 的内存使用情况

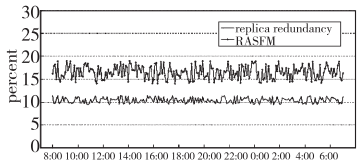

图 6 CPU 占用率对比

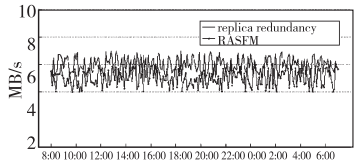

图 7 系统内部数据的传输流量对比

从图 4、5 可知,SFMAF 在内存占用,特别是内存 Cache 占用率上要大于使用副本冗余机制的系统。从图 6 可知,采用 SFMAF 文件系统的 CPU 占用率比使用副本冗余机制的文件系统的占用率要稍高一些。从图 7 可知,采用 SFMAF 文件系统的内部数据的传输流量要比采用副本冗余机制的少。

从上面图中可以看出,SFMAF 相对于副本冗余机制有以下几个不同点:a)系统需要更多的缓存资源存储用于纠删码计算的矩阵数据;b)SFMAF 中文件写入文件系统后会对其纠删码校验码数据块进行计算,计算量的增加加大了 CPU 的占用率;c)因为采用 SFMAF 文件系统的内部数据的传输流量要比采用副本冗余机制的少,说明 SFMAF 的副本量要小于使用副本冗余机制的系统。

综上所述,在 CPU 和内存使用率可接受范围内的增加上,SFMAF 的副本量少于使用副本冗余机制的系统,减少了系统内部数据的传输流量。

5 结束语

本文算法 SFMAF 对于常读取的文件采用副本冗余机制,对于不常读取的文件采用 RS 纠删码容错机制。SFMAF 有其自身的优势,但仍存在不足,进一步的工作主要有以下两个方面:

a)在不同的环境中,测试文件访问频率表的大小和参数 *num* 对算法的影响,为算法设置最优的表大小。

b)纠删码算法需要消耗的计算资源、内存资源较多,对纠删码算法改进可以提高本文算法的性能。

参考文献:

[1] 张建勋,古志民,郑超. 云计算研究进展综述[J]. 计算机应用研究,2010,27(2):429-433.

[2] 陈康,郑纬民. 云计算:系统实例与研究现状[J]. 软件学报,2009,20(5):1337-1448.

[3] ZAWODNY J. Yahoo! launches World's largest HadoopProduction application[EB/OL]. (2008-02-19). http://developer. yahoo. com/blogs/hadoop/posts/2008/02/yahoo-worlds-largest-production-hadoop/.

[4] GIBSON G, METER R. Network attached storage architecture [J]. Communications of the ACM, 2000, 43(11):37-45.

[5] 马玮骏,吴海佳,刘鹏. MassCloud 云存储系统构架及可靠性机制[J]. 河海大学学报:自然科学版,2011,39(3):348-354.

[6] Object storage developer documentation swift [EB/OL]. (2012-03-12). http://swift. openstack. org/.

[7] RIZZO L. On the feasibility of software FEC [EB/OL]. (1997-01-31). http://teamfrednet-mockups. googlecode. com/files/Rizzo-Feas-Soft-FEC-97. pdf.

[8] REED I S, SOLOMON G. Polynomial codes over certain finite fields [J]. Journal of the Society for Industrial and Applied Mathematics, 1960, 8(2):300-304.

[9] 罗象宏,舒继武. 存储系统中的纠删码研究综述[J]. 计算机研究与发展,2012,41(9):1-11.

[10] PLANK J S. A tutorial on reed-solomon coding for fault-tolerance in RAID-like systems [J]. Software Practice and Experience, 1997, 27(9):995-1012.

[11] 向茜,刘钊. 伽罗华域上代数运算的最简实现[J]. 电子科技大学学报,2000,29(1):5-8.